

A Two-Matrix View of a ReLU Layer

Instead of seeing a layer as a monolith, split it conceptually into:

- **A (feature / gating matrix)**
Learns hyperplanes that partition input space.
- **W (readout / content matrix)**
Learns how to combine the selected features into outputs.

The forward pass becomes:

- **A decides “where you are”** (which region of input space)
- **W decides “what to output there”**

Consider a standard layer:

$$y = W \sigma(Ax)$$

where $\sigma(\cdot)$ is ReLU.

Define a **gating vector**:

$$g(x) = \mathbf{1}[Ax > 0] \in \{0, 1\}^m$$

Then the hidden activation can be written as:

$$\sigma(Ax) = g(x) \odot (Ax)$$

So the output becomes:

$$y = W(g(x) \odot (Ax))$$

A as a Learnable Hash / Addressing System

Each row of **A** defines a hyperplane. Together they:

- Partition space into regions (polytopes)
- Produce a binary gating pattern (which ReLUs are on/off)

Interpretation:

- The gating pattern is a **data-dependent code**
- Nearby inputs tend to fall into similar regions → similar codes

So **A** is learning a **locality-sensitive hash**, but not fixed—it adapts during training.

Training effect on A:

- Moves hyperplanes to:
 - Align with important data boundaries
 - Separate inputs that need different outputs
 - Group inputs that should share behavior
-

W as Associative Memory

Given a gating pattern:

- Only a subset of columns/features are active
- **W combines them into the output**

Interpretation:

- Columns of **W** are **reusable memory fragments**
- The active set (chosen by A) determines which fragments are retrieved

Training effect on W:

- Stores associations between regions (codes) and outputs
 - Adjusts weights to reduce error for the currently active features
-

Training Dynamics: Division of Labor

A useful mental model:

- **W learns fast, local corrections**
 - “Given this current partition, how do I map to the right output?”
- **A learns slower, structural changes**
 - “Is this partition even the right one?”

This creates a feedback loop:

1. A produces a partition (a coding of inputs)
 2. W learns to fit outputs under that partition
 3. Gradients push A to *improve the partition* so W can do its job more easily
-

What “Better Partitions” Mean

A improves performance when it:

- **Separates conflicting associations**
 - Inputs needing different outputs should activate different features
 - **Clusters compatible inputs**
 - Inputs with similar outputs should share features
 - **Balances usage**
 - Avoids too many inputs sharing the same small set of features (reduces interference)
-

Capacity and Interference Revisited

From this viewpoint:

- If **A produces too-similar codes** for many inputs:
 - W must reuse the same columns → interference → noisy outputs
- If **A produces well-separated but local codes**:
 - W can store associations cleanly → better recall and generalization
- If **A over-fragments space**:
 - Too many tiny regions → poor generalization (memorization)

So training is implicitly balancing:

- **separation vs sharing**
 - **capacity vs generalization**
-

Connection to Earlier Viewpoints

This ties together:

- **Hashing / LSH**
 - A learns the hash that makes similar inputs collide usefully

- **Associative memory**
W stores and retrieves outputs based on those hashes
 - **Random feature models (as a limit case)**
If A is fixed random → only W learns (simpler, less adaptive)
-

Key Intuition

Training a ReLU network is a co-adaptation process:

- **A learns how to *address* the memory**
- **W learns what to *store* at those addresses**

Good performance comes from:

- A creating a **useful geometry of regions**
 - W efficiently **encoding associations within that geometry**
-

Takeaway

This two-matrix view turns a ReLU network into:

- **A learned partitioning system (A)**
- Coupled with a **distributed associative memory (W)**

It's a simple lens, but it explains:

- why feature learning matters,
 - how capacity limits arise,
 - and why geometry (not just weights) is central to performance.
-